KeyVM

Apr 24, 2023

Contents:

CHAPTER 1

Quickstart

Tested with Python 3.8.2

1.1 Download

```
git clone https://github.com/void4/keyvm.git
```

1.2 Installation

```
pip install -r requirements.txt
```

1.3 Usage

```
python main.py
```

CHAPTER 2

Architecture

2.1 General properties of KeyVM

- stack-based computational architecture - most operations take their inputs from a stack of previous results, and push their results onto it
- single-threaded - only one instruction is executed at a time
- deterministic - a copy of the program with the same input will always create the same results, down to the bit
- stateless, image-based - all data necessary to restore a process resides in the process image, which can be saved to and loaded from a file after execution ends

2.2 Access right model

What makes this architecture unique is that it introduces a new concept - *(access) rights* - represented by objects called keys.

Everything in this architecture is derived from two types of objects:

- Key: gives access to a Page. It is *impossible* to read from or write to a page you don't have the key to. It works just like real, physical keys and locks.
- Page: there are two types: DataPages (a contiguous sequence of bytes, used to store code and data) and KeyPages (a list of keys), both of dynamic size.

Keys to an existing Page cannot be constructed out of nothing - they cannot be forged - only the creator of a page receives its key.

Because keys can be copied and shared by those who already have them, more than one domain can have access to the same page, and which domain has which keys can change over time.

Instead of sending data to another domain, you just send a copy of the key that allows it to access the page the data is contained in. Now, data is not the thing that matters, but the rights to it.

Instead of a process, where every part of the program is allowed to access every other part, here a domain can construct a new domain, put some code into it and run it, with the complete assurance that it can only access the pages it was explicitly given access to - nothing else.

2.2.1 Domains

A domain is the KeyVM equivalent to a typical process.

In contemporary programming languages a process has a fixed code and data memory attached to (only) it. Here, a process is just a list of keys called a *Domain*.

It's just a KeyPage that contains Keys that point to other Pages that a process needs - the processes' code, data and stack as well as some more information (more at Data Structures).

A Domain can send other domains their DomainKey - a special key that allows others to send it a message.

Domains pass control to each other by invoking the CALL instruction with the DomainKey of the other domain and (optionally) another - a 'message' Key - which is copied to the called domains' KeyPage. The called Domain can then access everything (indirectly) referred to by key (a Page if it's a PageKey, or more keys if it's a KeyPageKey). In this implementation, only one domain is running at a time (single-threaded architecture).

2.2.2 Meters

MeterKeys account for resources, specifically computation time and memory limits. Domains have to own a time meter key in order to run the code they refer to. MeterKeys are organized in a tree hierarchy, each meter has a parent meter, up to the so called Prime Meters.

In order to allocate more pages, one needs a Memory MeterKey with sufficient resources. When a domain runs, the entire meter chain up to the prime meter is decreased at every execution step. When a meter runs out of time, its the controller of its parent (RESEARCH) receives control.

More info here:

- http://www.cap-lore.com/Agorics/Library/KeyKos/key370.html
- http://www.cap-lore.com/Agorics/Library/KeyKos/

2.3 Main actions

- create a new KeyPage or DataPage
- copy a Key (within the same or to another KeyPage)
- attenuate a Key (weaken its rights, e.g. PageKey -> PageReadKey)
- send a message and transfer control to another domain by calling the PageKey that refers to it
- create a new domain from a prepopulated KeyPage

2.3.1 Special instructions

CHAPTER 3

Assembler

The assembler accepts instructions in a tree format

`memwrite(memread(0,0),1,codelen())`

is expanded to

```
PUSH 0
PUSH 0
MEMREAD
PUSH 1
CODELEN
MEMWRITE
```

It doesn't yet check if the inputs/outputs match up.

CHAPTER 4

Data Structures

4.1 Domain

A Domain is a KeyPage, with the following structure:

CHAPTER 5

API

exception vm.**VMException**

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

v

Index

V